AutoMoe

Mar 27 2018

4th Progress Presentation

Lateef Adetona, Tavares Kidd, Jordan Lafontant, Collin Scott

Milestone Update

Mar	1	Write code for LCD display and compass	jordan/Collin
	2	Write code for GPS and navigation	jordan/Collin
	3	Write code for sensors and remaining functions	jordan/Collin
	4	Test code and make adjustments	jordan/Collin
Apr	1	Final Testing	Lateef

Updates (Hardware)

- Individual component testing has concluded.
- *Play video on ultrasonic distance sensor

Updates (Software)

- Arduino navigation code via peripherals

Updates (Arduino)

```
void processGPS(void)
{
  currentLat = convertDegMinToDecDeg(GPS.latitude);
  currentLong = convertDegMinToDecDeg(GPS.longitude);

  if (GPS.lat == 'S')            // make them signed
    currentLat = -currentLat;
  if (GPS.lon = 'W')
    currentLong = -currentLong;

  // update the course and distance to waypoint based on our new position
  distanceToWaypoint();
  courseToWaypoint();

}   // processGPS(void)
```

```
int distanceToWaypoint()
{

  float delta = radians(currentLong - targetLong);
  float sdlong = sin(delta);
  float cdlong = cos(delta);
  float lat1 = radians(currentLat);
  float lat2 = radians(targetLat);
  float slat1 = sin(lat1);
  float clat1 = cos(lat1);
  float slat2 = sin(lat2);
  float clat2 = cos(lat2);
  delta = (clat1 * slat2) - (slat1 * clat2 * cdlong);
  delta = sq(delta);
  delta += sq(clat2 * sdlong);
  delta = sqrt(delta);
  float denom = (slat1 * slat2) + (clat1 * clat2 * cdlong);
  delta = atan2(delta, denom);
  distanceToTarget =  delta * 6372795;

  // check to see if we have reached the current waypoint
  if (distanceToTarget <= WAYPOINT_DIST_TOLERANE)
    nextWaypoint();

  return distanceToTarget;
}  // distanceToWaypoint()
```

```
void calcDesiredTurn(void)
{
    // calculate where we need to turn to head to destination
    headingError = targetHeading - currentHeading;

    // adjust for compass wrap
    if (headingError < -180)
      headingError += 360;
    if (headingError > 180)
      headingError -= 360;

    // calculate which way to turn to intercept the targetHeading
    if (abs(headingError) <= HEADING_TOLERANCE)      // if within tolerance, don't turn
      turnDirection = straight;
    else if (headingError < 0)
      turnDirection = left;
    else if (headingError > 0)
      turnDirection = right;
    else
      turnDirection = straight;

}  // calcDesiredTurn()
```

```
int courseToWaypoint()
{
  float dlon = radians(targetLong-currentLong);
  float cLat = radians(currentLat);
  float tLat = radians(targetLat);
  float a1 = sin(dlon) * cos(tLat);
  float a2 = sin(cLat) * cos(tLat) * cos(dlon);
  a2 = cos(cLat) * sin(tLat) - a2;
  a2 = atan2(a1, a2);
  if (a2 < 0.0)
  {
    a2 += TWO_PI;
  }
  targetHeading = degrees(a2);
  return targetHeading;
}   // courseToWaypoint()
```

void moveAndAvoid(void)

```
void moveAndAvoid(void)
{

    if (sonarDistance >= SAFE_DISTANCE)       // no close objects in front of car
        {
          if (turnDirection == straight),+
            speed = FAST_SPEED;
          else
            speed = TURN_SPEED;
          driveMotor->setSpeed(speed);
          driveMotor->run(FORWARD);
          turnMotor->run(turnDirection);
          return;
        }

     if (sonarDistance > TURN_DISTANCE && sonarDistance < SAFE_DISTANCE)    // not
     yet time to turn, but slow down
       {
         if (turnDirection == straight)
           speed = NORMAL_SPEED;
         else
           {
              speed = TURN_SPEED;
              turnMotor->run(turnDirection);      // alraedy turning to navigate
           }
         driveMotor->setSpeed(speed);
         driveMotor->run(FORWARD);
         return;
       }

     if (sonarDistance <  TURN_DISTANCE && sonarDistance > STOP_DISTANCE)  //
     getting close, time to turn to avoid object
        {
          speed = SLOW_SPEED;
          driveMotor->setSpeed(speed);      // slow down
          driveMotor->run(FORWARD);
          switch (turnDirection)
          {
            case straight:                  // going straight currently, so start
            new turn
              {
                if (headingError <= 0)
                  turnDirection = left;
                else
                  turnDirection = right;
                turnMotor->run(turnDirection);  // turn in the new direction
                break;
              }

```

case left: // if already

```
case left:                          // if already turning left, try right
            {
              turnMotor->run(TURN_RIGHT);
              break;
            }
          case right:                       // if already turning right, try left
            {
              turnMotor->run(TURN_LEFT);
              break;
            }
         } // end SWITCH

        return;
      }
if (sonarDistance <  STOP_DISTANCE)          // too close, stop and back up
     {
       driveMotor->run(RELEASE);            // stop
       turnMotor->run(RELEASE);             // straighten up
       turnDirection = straight;
       driveMotor->setSpeed(NORMAL_SPEED);  // go back at higher speet
       driveMotor->run(BACKWARD);
       while (sonarDistance < TURN_DISTANCE)       // backup until we get safe
       clearance
         {
            if(GPS.parse(GPS.lastNMEA()) )
               processGPS();
            currentHeading = readCompass();    // get our current heading
            calcDesiredTurn();                 // calculate how we would
            optimatally turn, without regard to obstacles
            checkSonar();
            updateDisplay();
            delay(100);
         } // while (sonarDistance < TURN_DISTANCE)
       driveMotor->run(RELEASE);        // stop backing up
       return;
      } // end of IF TOO CLOSE

}   // moveAndAvoid()

```

// Steering/turning

```
// Steering/turning
enum directions {left = TURN_LEFT, right = TURN_RIGHT, straight = TURN_STRAIGHT} ;
directions turnDirection = straight;


// Object avoidance distances (in inches)
#define SAFE_DISTANCE 70
#define TURN_DISTANCE 40
#define STOP_DISTANCE 12


// Speeds (range: 0 - 255)
#define FAST_SPEED 150
#define NORMAL_SPEED 125
#define TURN_SPEED 100
#define SLOW_SPEED 75
int speed = NORMAL_SPEED;



// IR Receiver
#ifdef USE_IR
  #include "IRremote.h"            // IR remote
  #define IR_PIN 5
  IRrecv IR_receiver(IR_PIN);           // create instance of 'irrecv'
  decode_results IR_results;            // create instance of 'decode_results'
#endif


// IR result codes
#define IR_CODE_FORWARD 0x511DBB
#define IR_CODE_LEFT 0x52A3D41F
#define IR_CODE_OK 0xD7E84B1B
#define IR_CODE_RIGHT 0x20FE4DBB
#define IR_CODE_REVERSE 0xA3C8EDDB
#define IR_CODE_1 0xC101E57B
#define IR_CODE_2 0x97483BFB
#define IR_CODE_3 0xF0C41643
#define IR_CODE_4 0x9716BE3F
#define IR_CODE_5 0x3D9AE3F7
#define IR_CODE_6 0x6182021B
#define IR_CODE_7 0x8C22657B
#define IR_CODE_8 0x488F3CBB
#define IR_CODE_9 0x449E79F
#define IR_CODE_STAR 0x32C6FDF7
#define IR_CODE_0 0x1BC0157B
#define IR_CODE_HASHTAG 0x3EC3FC1B


//
// Interrupt is called once a millisecond, looks for any new GPS data, and stores it
SIGNAL(TIMER0_COMPA_vect)
{
  GPS.read();
}

```

// Implement an IR "kill switch" if selected in co

```
// Implement an IR "kill switch" if selected in configuration options
#ifdef USE_IR
  void checkKillSwitch(void)
  {
    if(IR_receiver.decode(&IR_results))     // check for manual "kill switch"
        {
          turnMotor->run(RELEASE);
          driveMotor->run(RELEASE);

          lcd.clear();
          lcd.print(F("Press to resume"));
          delay(1000);

          IR_receiver.resume();
          while(!IR_receiver.decode(&IR_results)) ;  // wait for key press
          IR_receiver.resume();  // get ready for any additional input
        }
  }  // checkKillSwitch()
#endif
```

Risk Management

- Battery may be insufficient to power entire breadboard & Arduino
  - Battery can only output a max of 5 volts via USB cable

Next Steps

- Build the circuit!
  - Have all components powered by a mobile battery
- Bluetooth connectivity from app to arduino
- Beta testing